Trajectory Clustering

Teemu Pulkkinen
• What can we gain from clustering trajectories?
• How can we determine if two trajectories are similar?
• What are the benefits and problems of the different similarity measures?
• How can we use similarity measures for clustering?
Introduction

• Proliferation of GPS devices and indoor positioning brings about opportunities for collecting *trajectories*.
• E.g.: vehicles, but also animals and even phenomena like hurricanes.
• It is safe to assume that objects that move in a similar way have similar guiding principles.
• If we can detect this similarity, it is likely that we can discover interesting trends.
Motivation

- Tracking animals and analyzing their trajectories, we can discover migratory behavior
- Understanding hurricane trajectories helps establish early warning systems
- Tracking customers in a supermarket carries several benefits for both researchers and retailers:
  - Find ”hot/coldspots” in the store; areas that suffer from congestion or are rarely visited
Motivation

- Detect common routes people take
  - Customers that move in similar patterns are likely looking for the same things, or are working under the same constraints
  - Plan displays accordingly

- Sometimes the motivation can even seem counter-intuitive:
  - Retailers want people to spend time in their stores, and expose them to shopping opportunities
  - A lot of shops reflect this in their layouts
Preliminaries

• Trajectories can be considered a special case of *time series*, in 3 dimensions
  • *Coordinates + time*
  • *Time series analysis* is a well-defined, and well-researched, topic
  • E.g. in marketing, this is used to analyze the development of stocks over time
  • Most common approach is to analyze the shape of the actual time series object
Time series objects vs. trajectories

- Prior research affords a lot of tools for comparing time series
- This translates nicely to the analysis of trajectories
- A popular approach is to consider the trajectory as a sequence of symbols
  - In other words, we can see it as a *string object*
- We can then compare two trajectories using approaches very similar to *edit distance* (or *Levenshtein distance*)
Similarity measures, features

- Several different approaches to comparing trajectories
- Most choose to focus on a specific set of problems
  - Pros and cons in all approaches
- Metricity
  - Distance measure that is metric is easier to use for indexing
    - Proper indexing speeds up clustering
  - Proofs of convergence, time complexity, etc. usually easier to justify
Metricity, in detail

• To be considered a metric, a distance measure has to satisfy 4 conditions:

• $D(x, y) \geq 0$, non-negativity
• $D(x, y) = 0$ iff $x = y$, identity of indiscernibles
• $D(x, y) = D(y, x)$, symmetry
• $D(x, z) \leq D(x, y) + D(y, z)$, triangle inequality
Indexing

- Previous lecture mentioned storing trajectory data in location databases
- A metric distance measure allows for efficient indexing of trajectories
  - We can make assumptions about distances since we know they satisfy metricity conditions
  - Searching is efficient since we can ignore parts of the data that is justifiably irrelevant
Indexing, example: R-trees

• A tree structure that is especially suitable for storing location data
  • Considered an extension of B-trees
  • Locations make up the leaves in the tree
  • Parents represent rectangular regions of locations in their children
    • Specifically: MBR:s (Minimum Bounding Rectangles)
  • Focus on minimizing the empty space a rectangle covers
  • Regions can be queried from the tree
    • If the region does not intersect the rectangle of a parent node, we know it will not intersect any of the children either
"Which restaurants are within 5 km from me?"

If the region we are looking for is within A, we only need to look through \{a,b,c,d\} (and their child nodes).
Similarity measures, more features

- **Completeness**
  - Most distance measures force a comparison between all elements in both trajectories
    - A significant difference in only one section might overshadow the similarity in others
  - Comparing only sections that are similar means we avoid outliers
- **Efficiency**
  - Dynamic programming approach is usually demanding
Similarity measures, more features

- **Time dilation**
  - Similar trends might take place over different periods of time
  - Factor out speed as a variable; consider the overall shape

- **Robustness**
  - Systematic noise (e.g. from measurements) might have a cumulative effect
  - Outliers might have a significant impact on the total distance
Similarity measures: Euclidean

- The simplest possible measure between two time-series objects
- Compare the values in the objects at the same time instance, $t_i$
  + Simple to implement, intuitive approach
  + Fastest approach
  - Naive approach means that no time dilation is taken into consideration
  - Any offset between objects has a cumulative effect
  - Noise is detrimental
Euclidean, visually
Dynamic programming

• The concept of building a solution to a problem by combining the solutions of its subproblems
• In this case, the results of previous calculations are reused in future steps
• A matrix is usually used for storing intermediate values, maintaining a running estimate of the distance
  • In addition to the total distance, this matrix contains the distances between all prefixes
• How the matrix is built and traversed is what separates most popular trajectory similarity measures
Edit distance as an example of dynamic programming

• Familiar from the field of *string processing*

• Define the distance between two sequences of symbols (strings) as the amount of edits needed to transform one into the other:
  • Insert
  • Delete
  • Transform

• The common approach to solving this distance is to use *dynamic programming*
The value for each element in the EDIT matrix is defined as follows:

\[
D_{EDIT}(A_i, B_j) = \begin{cases} 
EDIT(i - 1, j - 1) & \text{if } A_i = B_j \\
1 + \min(EDIT(i - 1, j - 1), \text{EDIT}(i, j - 1), \text{EDIT}(i - 1, j)) & \text{if } A_i \neq B_j 
\end{cases}
\]

In addition to the trivial case where either string is empty:

\[
D_{EDIT}(A, B) = \text{length}(A) \text{ if } \text{length}(B) = 0, \text{ and vice versa}
\]
Edit distance, example

\[ D_{EDIT}(bored, snore) = \]

\[
\begin{array}{cccccc}
  & s & n & o & r & e \\
 0 & 1 & 2 & 3 & 4 & 5 \\
b & 1 & 1 & 2 & 3 & 4 & 5 \\
o & 2 & 2 & 2 & 3 & 4 & 4 \\
r & 3 & 3 & 3 & 3 & 2 & 3 \\
e & 4 & 4 & 4 & 3 & 2 & 2 \\
d & 5 & 5 & 5 & 5 & 4 & 3 \\
\end{array}
\]

- \text{transform}
- \text{insert/delete}
- \text{same}

= 3
Similarity measures: DTW

- **Dynamic Time Warping**
- A dynamic programming approach to time series analysis
- Behaves like edit distance most of the time, but carries a dynamic penalty:
  - Instead of adding ”1” as a cost, consider the actual distance between the elements
  - Time differences are taken into account
  - Provides a better match than Euclidean measure
  - Mapping between all objects in both trajectories means every discrepancy is considered and added to the penalty
  - Not metric
DTW, more formally

DTW calculation nearly identical to edit distance:

\[ D_{DTW}(A_i, B_j) = \begin{cases} 
    DTW(i - 1, j - 1) & \text{if } A_i = B_j \\
    \text{cost} + \min(\text{DTW}(i - 1, j - 1), \text{DTW}(i, j - 1), \text{DTW}(i - 1, j)) & \text{if } A_i \neq B_j 
\end{cases} \]

where

\[ \text{cost} = d(A_i, B_j) \]
DTW, visually
Similarity measures: LCSS

- **Longest Common SubSequence**
- Inverts the problem: which parts of the trajectory are similar
- Similar dynamic programming approach
  - Increase similarity when elements match
  - ”Matching” can be defined by a distance threshold
- Ignores parts that don’t match
- Good with noise and outliers (because of the above)
- Allows for time distortion
- Not metric
LCSS, formally

LCSS in its traditional form is actually a similarity measure:

$$S_{LCSS}(A_i, B_j) = \begin{cases} 
\emptyset & \text{if } i = 0 \text{ or } j = 0 \\
1 + LCS(A_{i-1}, B_{j-1}) & \text{if } A_i = B_j \\
\max(LCS(A_i, B_{j-1}), LCS(A_{i-1}, B_j)) & \text{if } A_i \neq B_j 
\end{cases}$$

But we can easily express this as a distance (or dissimilarity) measure:

$$D_{LCSS}(A, B) = 1 - \frac{S_{LCSS}(A, B)}{\min(n,m)},$$

where $n$ and $m$ are the lengths of $A$ and $B$, respectively.
**LCSS, example**

\[ S_{LCSS}(bored, snore) = \]

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>r</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>b!=s</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>b!=n</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>o!=s</td>
<td>0</td>
<td>o=o</td>
<td>1</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>o!=r</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

→ **Different character:** \[ \text{max}([i-1,j],[i,j-1]) \]
→ **Same character**, add 1 to score

= 3
LCSS, visually

Outlier is ignored

Similarity = total length of common subsections
Similarity measures: others

- **EDR: Edit Distance on Real sequence**
  - Relaxes equality requirement
  + Good with noise
  + More accurate than LCSS
  - Not metric

- **ERP: Edit distance with Real Penalty**
  - Considers distance between elements like DTW, but compares them to fixed value instead of each other
  + Handles distortion
  + Metric
  - Not as good with noise
Clustering

- Once we have determined the similarity between trajectories, we can cluster them into meaningful groups.
- The intuition is that trajectories in a cluster will exhibit similar characteristics.
- Traditional clustering approaches often define a centroid around which objects are clustered.
  - Not always clear what this means in terms of trajectories.
- A common approach to clustering once the distance between all points is known is agglomerative (or hierarchical) clustering.
Agglomerative clustering

- Some approaches described in earlier lectures
- Start with every object in its own cluster
- Merge clusters that are "close enough" to each other (based on some criteria)
- Determine a cutoff threshold where clustering is considered complete
  - E.g. maximum/minimum/average distance between elements of each cluster (distance threshold)
  - Pre-defined limit for the number of clusters
Agglomerative clustering, example

Cutoff threshold

Increasing distance between clusters

A  B  C  D  E  F  G  H  I

Resulting clusters

\{A,B,C,D,E,F,G,H,I\}
\{A,B,C,(DE),F,G,H,I\}
\{A,(BC),(DEF),G,H,I\}
\{A,(BCDEF),G,H,I\}
K-medoids

- Related to the K-means approach from previous lectures
- Cluster points around a *medoid* instead of the center point of the elements
  - Usually the most centrally located object, i.e. the one with the smallest average distance to the rest of the cluster members
- More robust than K-means
  - Minimizes pairwise dissimilarities instead of sum of squared Euclidean distances
    - Outliers have lesser effect
- Has a well-defined "representative" of the cluster that is actually an object (trajectory) itself
K-medoids, example
TRACLUS

- **TRAjectory CLUStering**
- A trajectory clustering approach that considers *sub-trajectories*
  - Parts of trajectories might match even when the trajectory as a whole doesn’t
- Distance is measured between similar *segments*
- Trigonometric measures used for distance calculation

(see Lecture IX, slide 23)
TRACLUS

- The MDL principle is used to define *characteristic points* in the trajectory:
  - Balance between *preciseness*\(^1\) and *conciseness*\(^2\)
  - \(^1\) Difference between original trajectory and model should not be too large
  - \(^2\) To be beneficial, the model should be smaller than the trajectory it is modelling
- Segments between characteristic points then become the target for clustering
- Clustering performed using DBSCAN
  - Lines instead of points, but Epsilon neighborhoods etc. remain
TRACLUS, example

$MDL_{\text{Partition}} > MDL_{\text{No-Partition}}$

$MDL_{\text{Partition}} > MDL_{\text{No-Partition}}$

$S_i$

$S_{i+1}$

$S_{i+2}$

$S_{i+3}$

$S_{i+4} = S_j$

- Characteristic point

- Trajectory partition
1. Partition trajectories

2. Calculate weighted sum of distances between them

\[ d_{\perp} = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \quad d_{||} = \text{Min}(l_{|| 1}, l_{|| 2}) \]

\[ d_\theta = ||L_j|| \times \sin \theta \]

3. DBSCAN

4. Define *representative* trajectory
Summary

- Trajectories can exhibit trends when clustered together
  - Useful for analysis
- The distance between trajectories is usually calculated with a dynamic programming approach
  - Main difference between approaches is how they calculate the matrix
  - DTW: consider the distance between elements
  - LCSS: compare sections that are similar
Summary

- Hierarchical clustering can use a distance matrix to define clusters based on the distances between the elements in them.
- Some approaches consider partitions of trajectories.
  - TRACLUS combines MDL simplification with a modified version of DBSCAN.
Literature

• **ERP:**
  - Chen, L. & Ng, R.: *On the marriage of Lp-norms and edit distance*
    Proceedings of the Thirtieth international conference on Very large databases - Volume 30, VLDB Endowment, 2004, 792-803

• **EDR:**
  - Chen, L.; Özsu, M. T. & Oria, V.: *Robust and Fast Similarity Search for Moving Object Trajectories*

• **TRACLUS:**
  - Lee, J.-G.; Han, J. & Whang, K.-Y.: *Trajectory clustering: a partition-and-group framework*